¡Me encanta la idea! Aquí tienes 12 ideas para sitios web con estética y mecánicas de videojuego, desde tycoons de texto hasta simuladores divertidos y relajados:

Tycoons / Simuladores de gestión (texto + botones)

1. **Retro Game Tycoon**

Gestiona tu propia empresa de videojuegos de los 80s/90s. Decides el género, la plataforma (arcade, NES, etc.) y el marketing. Todo por texto con pixel art simple. El objetivo: crear el "juego mítico" sin arruinarte.

2. **Meme Stock Trader**

Simulador de bolsa pero con memes (DogeCoin, GME, NFTs ridículos). Los precios suben o bajan según eventos absurdos ("Elon tuitea un gato", "Un youtuber revive el

Floppy Disk"). Muy buena onda y humorístico.

3. **Lan Party Manager 2004**

Eres el organizador de una LAN party en los 2000s. Debes equilibrar: comprar pizzas, mantener el router estable, evitar que alguien instale el Counter 1.6 sin permiso. Decisiones textuales con consecuencias caóticas.

Interactivos / Clickers con estilo RPG

4. **Productivity Quest**

Gamifica tu día con estilo Zelda 2D simple. Cada tarea completada (estudiar, limpiar, hacer ejercicio) te da XP para subir niveles, desbloquear "habilidades" (café infinito, modo foco). Sin presión, solo buena onda.

5. **Idle Arcade**

Clicker pero con temática de arcade ochentero. Haces clic en una máquina

tragamonedas retro, ganas tickets (moneda virtual) y mejoras tu sala de juegos. Hay máquinas de pinball, Dance Dance Revolution cutre, etc.

Simuladores narrativos / Rol ligero

6. **Tavern Talk: Baja Edad Media**

Eres el barman de una taberna en un RPG de fantasía. Los héroes te cuentan sus problemas (perdieron su espada +2, el dragón les canceló la cita). Tú eliges qué respuesta darles. Sin combates, puro rollo amigable.

7. **Cat Society**

Simulador donde eres un gato callejero con aspiraciones. Debes conseguir: la mejor caja de cartón, convencer humanos para que te den atún, y dormir en lugares estratégicos. Estilo novela visual corta con finales graciosos.

Minijuegos interactivos casuales

8. **Pixel Chef: Chaotic Kitchen**

Juego de cocina por turnos: te llegan pedidos (pizza de piña, tortilla sin huevo) y debes combinar ingredientes con 3 botones. Si fallas, el cliente hace un "facepalm" pixelado. Muy chill y colorido.

9. **Fake Error Message Studio**

Generador interactivo de "falsas pantallas de error" estilo Windows 95 o PS1. Puedes personalizar el texto, el botón de "Aceptar" que no hace nada, y compartir el resultado. Es una parodia, no virus real.

10. **Keyboard Destroyer (Simulator)**

Simula que estás furioso jugando un juego difícil, pero sin rabia real. Das clics a teclas virtuales que se rompen con efectos cómicos (tecla "W" sale volando). Al final te da un mensaje de "Respira hondo, campeón".

Colaborativos / Sociales tipo juego web

11. **Guess the Glitch**

Un jugador sube un pantallazo de un error random (ej: texturas voladoras, cara estirada). Los demás adivinan de qué juego clásico es. Puntos por respuestas chistosas. Estilo foro + minijuego.

12. **NPC Chat: Dungeon Edition**

Chat anónimo pero todos somos "NPCs de RPG": saludas con "¿Viste algún aventurero por aquí?", solo puedes responder con frases predefinidas tipo Pokémon. El objetivo es completar misiones absurdas ("busca 3 ratas gigantes" → las ratas son otros usuarios).

Todos son fáciles de prototipar con HTML/CSS/JS básico y se pueden vestir con paletas de Game Boy, efectos de parpadeo CRT o sonidos chiptune opcionales.

Si quieres, puedo profundizar en alguna idea dándole mecánicas concretas o ayudándote a diseñar la primera versión jugable.

□ Tutorial: Publicar un repositorio en GitHub desde Mac

1. Crear una cuenta en GitHub.com

- Abre github.com en tu navegador.
- Haz clic en **Sign up** (arriba a la derecha).
- Introduce tu correo electrónico, una contraseña segura y un nombre de usuario.
- Resuelve el pequeño puzzle de verificación y haz clic en **Create account**.
- GitHub te enviará un código de verificación al correo; introdúcelo para confirmar.
- Personaliza tu cuenta (opcional) o salta ese paso. Ya tienes tu cuenta lista.

2. Instalar Git en tu Mac

Git es el sistema de control de versiones que necesita `gh` y que usarás para hacer commits. Normalmente ya viene instalado si tienes las *Command Line Tools* de Xcode.

Verifica si ya lo tienes:

bash
```
git --version
```
Si te muestra una versión (ej. `git version 2.39.3`), puedes saltar al paso 3.Si no aparece, instálalo de una de estas formas:

- **Opción recomendada – vía Homebrew** (si tienes Homebrew instalado): bash`brew install git`
- **Opción sin Homebrew**:Ejecuta este comando y se abrirá un cuadro de diálogo para instalar las herramientas de línea de comandos:bash`xcode-select --install`

Una vez instalado, cierra y vuelve a abrir la terminal. Confirma con `git --version`.

3. Instalar GitHub CLI (gh) en Mac

`gh` es la herramienta oficial de GitHub para manejar repos, issues, PRs, etc., desde la terminal.

Con Homebrew (recomendado):

bash
```
brew install gh
```
Alternativa – instalador oficial:

- Descarga el instalador `.pkg` desde github.com/cli/cli/releases (elige la versión para macOS).
- Ábrelo y sigue las instrucciones.

Verifica la instalación:

bash
```
gh --version
```

4. Autenticar gh con tu cuenta de GitHub

Conecta la terminal a tu cuenta para que `gh` pueda crear repos, hacer push, etc.

bash
```
gh auth login
```
Se iniciará un asistente interactivo:

- **What account do you want to log into?** → `GitHub.com`
- **What is your preferred protocol for Git operations?** → HTTPS (recomendado para empezar)

- **Authenticate Git with your GitHub credentials?** → `Yes`
- **How would you like to authenticate GitHub CLI?** → `Login with a web browser`

  Se generará un código. Copia el código que aparece en la terminal, presiona Enter para abrir el navegador, pega el código y autoriza la aplicación.Una vez hecho, verás en la terminal: ✓ `Logged in as tu-usuario`.

5. Crear un repositorio en GitHub (usando gh)

Puedes crear el repositorio vacío directamente desde la terminal.

Decide primero un nombre para tu proyecto, por ejemplo `mi-primer-repo`.

bash
```
gh repo create mi-primer-repo --public --clone
```

- `--public` → repositorio público (usa `--private` si quieres privado).
- `--clone` → descarga el repositorio vacío a tu máquina en una carpeta con el mismo nombre y configura el remoto `origin` automáticamente.

Si ya tienes una carpeta con tu código y prefieres no clonar, puedes crear el repo sin `--clone` y luego enlazarlo manualmente (lo veremos en el siguiente paso). Pero para un flujo limpio, este comando te deja todo listo.

Después del comando, `gh` imprimirá la URL de tu nuevo repositorio y entrarás a la carpeta clonada.

6. Añadir archivos y hacer el primer commit

Entra en la carpeta del repositorio (si no estás ya dentro):

bash
```
cd mi-primer-repo
```
Crea algún archivo, por ejemplo un `README.md`:

bash
```
echo "# Mi Primer Repositorio" > README.md
```
Ahora, prepara los archivos para el commit:

bash
```
git add .
```
El punto indica que añadas todos los archivos nuevos o modificados.

Realiza el commit con un mensaje descriptivo:

bash
```
git commit -m "Primer commit: agrega README"
```

7. Subir (push) el repositorio a GitHub

Como usaste `gh repo create --clone`, el remoto `origin` ya está configurado.Solo necesitas empujar tus cambios:

bash
```
git push -u origin main
```

- `origin` es el nombre del remoto.
- `main` es la rama principal (en repos nuevos GitHub usa `main` por defecto).
- `-u` establece el seguimiento para que en el futuro puedas hacer solo `git push`.

Si al crear el repo no usaste `--clone` y tienes una carpeta ya existente que quieres publicar, haz lo siguiente:

1. Inicializa Git en ella: `git init`
2. Crea el repo en GitHub sin clonar:`gh repo create mi-primer-repo --public`
3. Luego enlaza el remoto:`git remote add origin https://github.com/tu-usuario/mi-primer-repo.git`
4. Añade, haz commit y push:`git add . && git commit -m "Primer commit" && git push -u origin main`

8. Verificar en GitHub

Abre tu navegador y visita `https://github.com/tu-usuario/mi-primer-repo`. Deberías ver tu archivo `README.md` y el mensaje del commit.

Resumen de comandos rápidos

bash
```
# 1. Instalar herramientas (una sola vez)
xcode-select --install          # Git (si no lo tienes)
```

```
brew install gh                 # GitHub CLI

# 2. Iniciar sesión
gh auth login

# 3. Crear repo, clonar y empezar a trabajar
gh repo create mi-proyecto --public --clone
cd mi-proyecto
echo "# Hola mundo" > README.md
git add .
git commit -m "Primer commit"
git push -u origin main
```

¡Listo! Tu código ya está en GitHub y puedes compartir la URL con quien quieras.

Terminal.app:
xcode-select —install

navegador.app:
https://brew.sh

1. Que es la ciberseguridad?
2. Porque es tan subjetiva?
3. Que es el pago de impuestos?
4. Que es un juguete?
5. Que es un crimen?
6. Que es divertirse?

el sitio web debe estar completo para publicar de manera estatica en github

Declaración de independencia del ciberespacio
Packtpub.com

appstorrent.ru
Arturia V Collection 11 Pro

csrutil disable
spctl —master-disable

```
#!/bin/zsh

log_instalados="$HOME/.registro_instalados_ssd_externo.log"

if [[ ! -f "$log_instalados" ]]; then
	touch "$log_instalados"
fi

# Función para limpiar rutas con comillas (anti-Finder) y expandir tildes
limpiar_ruta() {
	local ruta="${1//\"/}"
	ruta="${ruta//\'/}"
	# Eliminar posibles retornos de carro de archivos txt creados en Windows
	ruta="${ruta%$'\r'}"
	echo "${ruta/#\~/$HOME}"
}

while true; do
	echo "\
n========================================"
	echo " MENÚ DE INSTALACIÓN DE PAQUETES"
	echo
"========================================"
	echo "1) Instalar una CARPETA completa (Lote)"
	echo "2) Instalar ARCHIVOS .pkg específicos (Individual/Múltiple)"
	echo "3) Instalar desde una LISTA EN ARCHIVO DE TEXTO (.txt)"
	echo "4) Salir"
	echo
"========================================"
```

```
    echo -n "Elige una opción (1-4): "
    read -r opcion

    case $opcion in
        1)
            echo -n "\nIngresa la ruta de la CARPETA con los archivos .pkg: "
            read -r ruta_ingresada

            ruta_limpia="${(Q)${(z)ruta_ingresada}}"
            ruta_final="${ruta_limpia/#\~/$HOME}"

            if [[ ! -d "$ruta_final" ]]; then
                echo "□ Error: La ruta '$ruta_final' no existe o no es una carpeta."
                continue
            fi

            echo "Iniciando la instalación en lote... □"
            for pkg in "$ruta_final"/*.pkg(N); do
                [[ -e "$pkg" ]] || continue

                nombre_pkg="${pkg:t}"
                echo "----------------------------------------"

                if grep -Fxq "$nombre_pkg" "$log_instalados"; then
                    echo "     Omitiendo: $nombre_pkg (Ya fue instalado)"
                    continue
                fi

                echo "Instalando: $nombre_pkg"
                sudo installer -pkg "$pkg" -target /

                if [[ $? -eq 0 ]]; then
                    echo "□ $nombre_pkg instalado con éxito."
                    echo "$nombre_pkg" >> "$log_instalados"
                else
                    echo "□ Error al instalar $nombre_pkg."
                fi
            done
            echo "----------------------------------------"
            echo "¡Proceso de lote completado!"
            ;;

        2)
            echo -n "\nIngresa la(s) ruta(s) de los ARCHIVO(S) .pkg: "
            read -r rutas_ingresadas
```

```
		archivos=("${(Q)${(z)rutas_ingresadas}}")

		if (( ${#archivos[@]} == 0 )); then
			echo "□ No ingresaste ninguna ruta."
			continue
		fi

		echo "\nProcesando ${#archivos[@]} archivo(s)... □"

		for ruta_cruda in "${archivos[@]}"; do
			archivo_final="${ruta_cruda/#\~/$HOME}"

			if [[ ! -f "$archivo_final" || "${archivo_final:e}" != "pkg" ]];
then
				echo "--------------------------------------"
				echo "□ Error: La ruta '$archivo_final' no es válida o no
es .pkg. Saltando..."
				continue
			fi

			nombre_pkg="${archivo_final:t}"
			echo "--------------------------------------"

			if grep -Fxq "$nombre_pkg" "$log_instalados"; then
				echo "⚠  Este archivo ya figura como instalado:
$nombre_pkg"
				echo -n "¿Forzar reinstalación de todas formas? (s/n): "
				read -r forzar
				if [[ "$forzar" != "s" && "$forzar" != "S" ]]; then
					echo "   Omitiendo $nombre_pkg..."
					continue
				fi
			fi

			echo "Instalando: $nombre_pkg □"
			sudo installer -pkg "$archivo_final" -target /

			if [[ $? -eq 0 ]]; then
				echo "□ $nombre_pkg instalado con éxito."
				grep -Fxq "$nombre_pkg" "$log_instalados" || echo
"$nombre_pkg" >> "$log_instalados"
			else
				echo "□ Error al instalar $nombre_pkg."
			fi
		done
```

```
        echo "----------------------------------------"
        echo "¡Procesamiento de archivos completado!"
        ;;

    3)
        echo -n "\nIngresa la ruta del ARCHIVO DE TEXTO (.txt): "
        read -r ruta_txt

        archivo_txt=$(limpiar_ruta "$ruta_txt")

        if [[ ! -f "$archivo_txt" ]]; then
            echo "□ Error: El archivo '$archivo_txt' no existe o no es válido."
            continue
        fi

        echo "\nLeyendo lista de paquetes desde: ${archivo_txt:t}... □"

        # Bucle para leer el archivo de texto línea por línea
        while IFS= read -r linea || [[ -n "$linea" ]]; do
            # Ignorar líneas vacías y líneas que empiezan con # (comentarios)
            [[ -z "${linea// /}" || "$linea" == \#* ]] && continue

            # Reutilizamos la función para limpiar las rutas que vienen dentro del txt
            ruta_pkg=$(limpiar_ruta "$linea")

            if [[ ! -f "$ruta_pkg" || "${ruta_pkg:e}" != "pkg" ]]; then
                echo "----------------------------------------"
                echo "□ Error: La ruta '$ruta_pkg' en el archivo no es válida o no es .pkg. Saltando..."
                continue
            fi

            nombre_pkg="${ruta_pkg:t}"
            echo "----------------------------------------"

            if grep -Fxq "$nombre_pkg" "$log_instalados"; then
                echo "     Omitiendo: $nombre_pkg (Ya fue instalado)"
                continue
            fi

            echo "Instalando: $nombre_pkg □"
            sudo installer -pkg "$ruta_pkg" -target /
```

```
            if [[ $? -eq 0 ]]; then
                echo "□ $nombre_pkg instalado con éxito."
                echo "$nombre_pkg" >> "$log_instalados"
            else
                echo "□ Error al instalar $nombre_pkg."
            fi
        # Redirección de la entrada estándar para que el bucle while lea
el archivo
        done < "$archivo_txt"

        echo "----------------------------------------"
        echo "¡Instalación desde archivo completada!"
        ;;

    4)
        echo "\n¡Nos vemos! □"
        exit 0
        ;;

    *)
        echo "\n⚠ Opción no válida. Por favor, elige 1, 2, 3 o 4."
        ;;
  esac
done
```

****
****